DOCUMENT RESUME

ED 149 732 IR 005 515

AUTHOR	Aronson, Jules
TITLE	Data Compression--A Comparison of Methods. Computer Science and Technology.
INSTITUTION	National Bureau of Standards (DCC), Washington, D.C. Inst. for Computer Sciences and Technology.
REPORT NO	NBS-SP-500-12
PUB DATE	Jun 77
NOTE	37p.; Some pages may be marginally legible due to print quality
AVAILABLE FROM	Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 (C13.10:500-12, $1.50)
EDRS PRICE	MF-$0.83 HC-$2.06 Plus Postage.
DESCRIPTORS	*Comparative Analysis; Computer Storage Devices; Cost Effectiveness; Data Bases; *Data Processing; Federal Government; Guidelines; Information Scientists; *Information Storage; Information Theory; Standards
IDENTIFIERS	Computer Software; *Data Compression

ABSTRACT

This report delineates the theory and terminology of data compression. It surveys four data compression methods--null suppression, pattern substitution, statistical encoding, and telemetry compression--and relates them to a standard statistical coding problem, i.e., the noiseless coding problem. The well defined solution to that problem can serve as a standard on which to base the effectiveness of data compression methods. The simple measure described for calculating the effectiveness of a data compression method is based on the characterization of the solution to the noiseless coding problem. Finally, guidelines are stated concerning the relevance of data compression to data processing applications. (Author/DAG)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED149732

COMPUTER SCIENCE & TECHNOLOGY:

Data Compression — A Comparison of Methods

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Jules Aronson

Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

IR005 515

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, Secretary
Dr. Sidney Harman, Under Secretary
Jordan J. Baruch, Assistant Secretary for Science and Technology
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Acting Director

Issued June 1977

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-12
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-12,39 pages (June 1977)
CODEN: XNBSAV

Library of Congress Cataloging in Publication Data

Aronson, Jules.
Data compression — a comparison of methods.
(Computer science & technology) (National Bureau of Standards special publication ; 500-12)
Bibliography: p.
Supt. of Docs. no.: C13.10:500-12
1. Data compression (Computer science) 2. Coding theory. I. Title. II. Series. III. Series: United States. National Bureau of Standards. Special publication ; 500-12.
QC100-U57 no. 500-12 [QA76.9.D33] 602'.1s [00.16'425] 77-608132

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1977

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 - Price $1.50
Stock No. 003-003-01797-3

TABLE OF CONTENTS

Page

1. Introduction 1

2. Survey of Data Compression Techniques 3

   2.1 Null Suppression 3

   2.2 Pattern Substitution 5

   2.3 Statistical Encoding 9

   2.4 Telemetry Compression 11

3. Analysis of Data Compression 12

   3.1 Noiseless Coding Problem 15

      3.1.1 Uniquely Decipherable Codes 16
      3.1.2 Optimal Codes 17

   3.2 Realization of Optimal Codes 18

   3.3 Synthesis of the Huffman Code 21

4. CONCLUSIONS 27

5. BIBLIOGRAPHY 30

Acknowledgments

I wish to acknowledge the help furnished by Beatrice Marron and Dennis W. Fife. With the encouragement and assistance of both, but especially Ms. Marron, the ideas and style of the paper were developed.

Data Compression - A Comparison of Methods.

Jules P. Aronson

One important factor in system design and in the design of software is the cost of storing data. Methods that reduce storage space can, besides reducing storage cost, be a critical factor in whether or not a specific application can be implemented. This paper surveys data compression methods and relates them to a standard statistical coding problem - the noiseless coding problem. The well defined solution to that problem can serve as a standard on which to base the effectiveness of data compression methods. A simple measure, based on the characterization of the solution to the noiseless coding problem, is stated through which the effectiveness of a data compression method can be calculated. Finally, guidelines are stated concerning the relevance of data compression to data processing applications.

Key words: Coding; Coding Theory; Computer Storage; Data Compaction; Data Compression; Data Elements; Data Management; Data Processing; Information Management; Information Theory.

1. Introduction

The purpose of this report is to assist Federal Agencies in developing data element standards that are both compatible within the Federal government and economical. Specifically, this report responds to the GAO recommendations that the Department of Commerce "... issue policy, delineating accepted theory and terminology, and provide for preparation of guidelines, methodology, and criteria to be followed by agencies in their standards efforts"*. This report delineates the theory and terminology of data compression and surveys classes of data compression techniques.

---

* GAO report B-115369; Emphasis Needed On Government's Efforts To Standardize Data Elements And Codes For Computer Systems; May 16, 1974; p33.

Data element standards activities in the past have been concerned with abbreviations or codes for specific terms, such as the names of countries, metropolitan areas, and states. The purpose of such representations has been to reduce the space necessary to store such terms, while maintaining the ability to reproduce the terms from the representations. While each representation in a given class is unique, inter class uniqueness is not necessarily maintained. For example, the standard abbreviation for CALIFORNIA is CA (1), but the abbreviation for CANADA is also CA (2). The use of standard codes creates similar problems. The code for the geographical area of Alameda County, California is 06001 (3). while that for the standard metropolitan statistical area of Augusta Georgia is 0600 (4). To distinguish between these two codes, whenever they occur in the same file, is complicated and sometimes impossible, since these codes violate a coding principle that one code not be a prefix of another (5). The decoding of the above two codes involves the inefficient process of backtracking through the message stream after it has been received.

The reduction in storage, effected by the use of data representations, is not as great as the reduction that can be accomplished by the use of uniform and systematic techniques of data compression. This report describes methods which uniformly compress the data, rather than a select set of terms. These methods may be used to replace standard representations or may be applied to data in which some terms are already so represented. These methods could reduce the high cost of computer operations by eliminating unnecessary incompatibilities in the representation of data and by reducing the cost of storing the data.

The cost of storing data is a very significant part of the total computer system cost. This cost is composed of the direct charges for the storage media, such as disk devices, as well as the costs of transfering the data to and from local and remote storage devices. The latter costs are in turn composed of the costs of the data channels and, for remotely stored data, the network, both of which must have sufficient bandwidth to transmit the data . Data compression results in cost savings by reducing the amount of storage required to store data files. In addition, data

---

(1) Nat. Bur. Stand., Fed. Info. Process. Stand. Publ. (FIPS PUB) 5-1
(2) FIPS PUB 10-1
(3) FIPS PUB 6-2
(4) FIPS PUB 8-4
(5) see section 3.1.1

compression methods may enable more efficient information retrieval operations as well as more economical transmission of large amounts of data over computer networks. There are several types of data compression techniques which range from the suppression of null characters to pattern substitution and statistical coding.

In this report several types of data compression techniques are discussed along with descriptions of some of their implementations. Then, the data compression problem is analyzed with respect to a classification of compression schemes in terms of the functional attributes of domain, range, and operation. In addition, concepts from information theory are introduced, in part 3, to give the reader a perspective from which to clarify and measure the performance of compression techniques. From information theory the compression problem may be seen as an aspect of the more general noiseless coding problem. The mathematical portions of part 3 may be skipped without seriously affecting the meaning of this report. Finally, some criteria for the selection of techniques are discussed with regard to the form and application of the data structure.

2. Survey of Data Compression Techniques

2.1 Null Suppression

Null suppression techniques encompass those methods which suppress zeros, blanks, or both. This type of compression could be called the de facto standard method for compressing data files. It takes advantage of the prevalence of blanks and zeros in some data files, and is easy and economical to implement. Null suppression may not, however, achieve as high degree of compression ratio as some other techniques. Its obvious application is to card image data records which formed the basic data structure of many of the earlier data management systems.

One way of implementing null suppression is through the use of a bit map in which a one indicates a non-null data item and a zero indicates a null item. This method is applicable to data files having fixed size units, such as words or bytes. Figure 1 illustrates the method where a bit map is appended in the front of a collection of items. Units containing all nulls are dropped from the collection and the bit which corresponds to such units is set to zero.

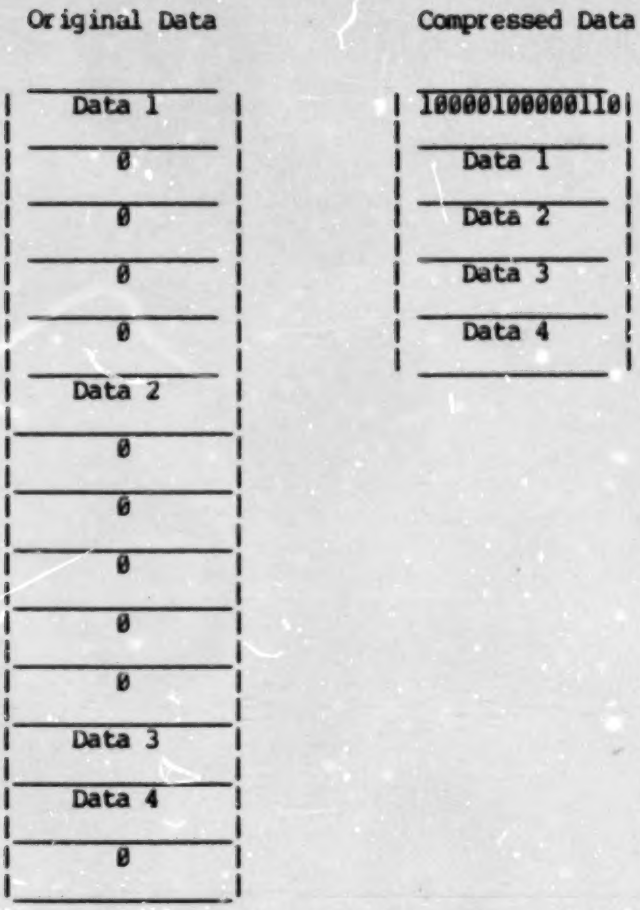


Figure 1 Zero Suppression Using a Bit Map

Another way to implement null suppression is the run length technique shown in figure 2. A special character is inserted to indicate a run of nulls. Following that character is a number to indicate the length of the run. The choice of the special character depends on the code used to represent the data. For codes such as ASCII or EBCDIC a good choice is one of the characters which does not occur in the data, of which there are many in these codes. If the character set contains no unused characters, such as in the six-bit codes, the technique may still be used by selecting an infrequently used character and doubling it when it occurs as part of the data.

Original Data: Item A10000X02500000N[illegible]COST

Compressed Data: Item A1$4X025$5N%5COST

Figure 2 Run Length Coding

2.2 Pattern Substitution

The run length technique is a primitive form of a class of techniques known as pattern substitution, in which codes are substituted for specific character patterns. Data files often contain repeating patterns, such as illustrated in figure 3. These may include numeric and alphabetic information combined with or in addition to null characters.

Original Data:

AE10004MFQ00000F320006BCX4
AE20000DBF0000[illegible]9000BCX1
AE30002RBA00000F30[illegible]214BCX7

Pattern Table:

AE = #
000 = $
00000F3 = %
BCX = @

Compressed Data

#1$4MFQ%2$6@4
#2$0DBF%$0$@1
#3$2RBA%0121 4@7

Figure 3 Pattern Substitution

A pattern table may be constructed either in advance or during the compression of the data. The table may be transmitted with the data or stored as a permanent part of the compressor and decompressor. In the method of De Main, Kloss, and Marron the pattern is stored with the data, while in the method of Snyderman and Hunt* the pattern is stored in the compressor and decompressor. As in null suppression,

---

*See reference 23

the code for the pattern is represented by unused characters from the character set.

The statistical properties of the patterns may be advantageously used to increase the efficiency of the compression. In the method of Snyderman and Hunt, even though trial and error was used to select the patterns, the resultant patterns were 168 of some of the most frequently occurring pairs of characters in their textual data files. The frequency of pairs of characters is further exploited by Jewell who chose 190 of the most frequently occurring pairs as candidates for substitution.

The compression method of Snyderman and Hunt and that of Jewell involve substituting single character codes for specific pairs of characters. They differ primarily in the way the pairs of characters are selected, and secondarily in the selection of the substitution code.

In the method of Snyderman and Hunt two lists of characters are selected based partly on their frequency of occurrence in English text. The first list, called the "master characters", is a subset of the second list called the "combining characters". In the example given by the authors there are eight master characters (blank,A,E,I,O,N,T,U) and 21 combining characters (blank,A,B,C,D,E,F,G,H,I,L,M,N,O,P, R,S,T,U,V,W).

The first step of the compaction process involves translating each character to a hexadecimal code between 00 and 41 leaving 190 contiguous codes at the end, 42 through FF, for the substitution codes. Next, each translated character is tested, in turn, to determine if it is a master character. If it is not such, then it is output as it is; otherwise, it is used as a possible first character of a pair. When a master character has been found, the next character in the input string is tested to determine if it is a combining character. If it is, then the code for the pair is calculated and replaces both of the input characters. If the next character is not a combining character then the translated hexadecimal representations for both are each moved to the output stream. Figure 4 contains a table of the compacted code, using this scheme.

COMPACTED CODE

Master Characters		Combining Characters		Noncombining Characters						Combined Pairs			
Char	Base Value	Char	Hex Code	Char	Hex Code	Char	Hex Code	Char	Hex Code	Char	Hex Code	Char	Hex Code
∅	58	∅	00	J	15	q	2B	<	41	∅∅	58	A∅	6D
A	6D	A	01	K	16	r	2C	(	42	∅A	59	AA	6E
E	82	B	02	Q	17	s	2D	+	43	∅B	5A	AB	6F
I	97	C	03	X	18	t	2E	&	44	∅C	5B	AC	70
O	AC	D	04	Y	19	u	2F	!	45	∅D	5C	..	..
N	C1	E	05	Z	1A	v	30	$	46	∅E	5D	AW	81
T	D6	F	06	a	1B	w	31	*	47	∅F	5E	E∅	82
U	EB	G	07	b	1C	x	32	)	48	∅G	5F	EA	83
		H	08	c	1D	y	33	;	49	∅H	60	..	..
		I	09	d	1E	z	34	-	4A	∅I	61	EW	96
		L	0A	e	1F	0	35	/	4B	∅L	62	I∅	97
		M	0B	f	20	1	36	,	4C	∅M	63	..	..
		N	0C	g	21	2	37	%	4D	∅N	64	O∅	AC
		O	0D	h	22	3	38	_	4E	∅O	65	..	..
		P	0E	i	23	4	39	>	4F	∅P	66	N∅	C1
		R	0F	j	24	5	3A	?	50	∅R	67	..	..
		S	10	k	25	6	3B	:	51	∅S	68	T∅	D6
		T	11	l	26	7	3C	#	52	∅T	69	..	..
		U	12	m	27	8	3D	@	53	∅U	6A	U∅	EB
		V	13	n	28	9	3E	'	54	∅V	6B	..	..
		W	14	o	29		3F	=	55	∅W	6C	VW	FF
				p	2A	.	40	"	56				
								<	57				

(in the above ∅ = blank)
Figure 4

Using the technique described, the Science Information Exchange compacted the text portion of a 200,000 record on-line file from an average of 851 to 553 characters per record, a decrease of 35 percent. Using an IBM 360/40 the compression takes 73 ms. for 1000 characters while expansion takes only 65 ms. The extent to which the decrease was due to null suppression can not be determined from the authors' report. Such a determination would be necessary before an accurate comparison between methods can be made.

The method of Jewell takes into account the full 190 most frequently occurring character pairs in his sample, thus taking advantage of the availability of the 190 unused codes in an 8-bit representation. Figure 5, compiled by Jewell, is a 2-character frequency distribution of the 25 most frequently occurring pairs in a sample of text. The 190 pairs are entered into a table which forms a semi-permanent part of the compaction process. The first step of the process involves shifting the first two characters of the input stream into a register. If this pair occurs in the combination table then a code is substituted for the pair. The code is the address of the pair in the table. Two new characters are then entered and the process resumes as in the beginning. If the input pair is not in the table then the first character of that pair is translated to a value greater then hexadecimal BD (which equals 190, the length of the table) and sent to the output stream. One new character is shifted in with the remaining second character and the process resumes.

Rank	Combination	Occurrences	Occurrences per Thousand
1	E∅	328	26.89
2	∅T	292	23.94
3	TH	249	20.41
4	∅A	244	20.00
5	S∅	217	17.79
6	RE	200	16.40
7	IN	197	16.15
8	HE	183	15.00
9	ER	171	14.02
10	∅I	156	12.79
11	∅O	153	12.54
12	N∅	152	12.46
13	ES	138	12.13
14	∅b	141	11.56
15	ON	140	11.48
16	T∅	137	11.23
17	TI	137	11.23
18	AN	133	10.90
19	D∅	133	10.90
20	AT	119	9.76
21	TE	114	9.35
22	∅C	113	9.26
23	∅S	113	9.26
24	OR	112	9.18
25	R∅	109	8.94

Partial results of a 2-character frequency test
The text size is 12198 characters
Figure 5.

2.3 Statistical Encoding

Statistical encoding is another class of data compression methods which may be used by itself or combined with a pattern substitution technique. Statistical encoding takes advantage of the frequency distribution of characters so that short representations are used for characters that occur frequently, and longer representations are used for characters that occur less frequently. When combined with pattern substitution, short representation may be used for

some frequently occurring pairs or other groups of characters. Morse code, for example, uses short code groups for the common letters, and longer code groups of the others.

When binary ones and zeros are used to represent a message in variable length codes, there must be a way to tell where one character or pattern ends and the other begins. This can be done if the code has the prefix property, which means that no short code group is duplicated as the beginning of a longer group. Huffman codes have the prefix quality and in addition are minimum redundancy codes, that is they are optimal in the sense that data encoded in these codes could not be expressed in fewer bits.

Figure 6 shows the combinatorial techniques used to form Huffman codes. The characters, listed in descending order of frequency of occurrence, are assigned a sequence of bits to form codes as follows. The two groups with the smallest frequencies are selected and a zero bit is assigned to one and a one bit is assigned to the other. These values will ultimately be the value of the right most bit of the Huffman code. In this case, the right most bit of A is 1, while that of B is 0, but the values of the bit assignments could have been interchanged. Next, the two groups, A and B, are then treated as if they were but one group, represented by BA, and will be assigned a specific value in the second bit position. In this way both A and B receive the same assignment in the second bit position. The above process is now repeated on the list E,T,4,BA, where BA represents groups A and B, and has frequency of 10%. The two least frequently occurring groups, represented by 4 and BA, are selected, and a zero bit is assigned to character 4 and a one bit is assigned to BA. These values will be the values of the second bit from the right of the Huffman code. The partial code assembled up to this point is represented in the step 2 column of Figure 6. In each of steps 3 and 4 the process is repeated, each time forming a new list by identifying the two elements of the previous list which had just been assigned values, and then assigning zero and a one bit to the two least frequently occurring elements of the new list. In this example, messages written in the Huffman codes require only 1.7 bits per character on the average, whereas three bits would be required in the fixed length representations. The synthesis of Huffman codes will be discussed in greater detail in the next section.

Character	Frequency	step 1	step 2	step 3	Huffman Code step 4
E	60 %				0
T	20 %			0	10
4	10 %		0	10	110
B	6 %	0	10	110	1110
A	4 %	1	11	111	1111

Figure 6 Formation of Huffman Code

2.4 Telemetry Compression

Telemetry compression techniques are not applicable to most data files. In telemetry, a sensing device records measurements at regular intervals. The measurements are then transmitted to a more central location for further processing. Compression is applied prior to transmission to reduce the total amount of data to be transmitted. Telemetry data is generally a sequence of numeric fields. In the sequence there are subsequences or runs of numeric fields with values that vary only slightly from each other. Compression is achieved by coding each field, other than the first, with the incremental difference between it and the preceding field, provided the absolute value of the increment is less than some pre-determined value. Otherwise, the field is represented as it is with some escape character to indicate that the particular field is not coded. The conditions that make the incremental coding technique effective, the existence of long runs of similarly valued fields, do not exit in most data files.

3. Analysis of Data Compression

Data compression may be represented as the application of some function to elements of the data base. If we let x be a specified element of the data base, then the compression of x is y=f(x).

Here, x, the element of the data base, may be a string of one or more bits, bytes, characters, pairs or n-tuples of characters, words, or text fragments. f is a function that maps the element x into some other element y. The domain of a function is that set upon which the function operates, while the range is that set whose elements are the results of the function operation. The different compression techniques may be characterized by the choice of the domain, range and the operation of the function f.

Usually f is invertible, which means that the original data may be recovered from the compressed data. However, in some applications, a non- invertible choice of f may be advantageous. For example, when the data base to be compressed consists of record identification keys, only an abbreviated form of each key may be necessary to retrieve each record. In that case a non-invertible compression technique that removes some of the information from each key would generate a more compressed key file than one that was invertible.

In the method of Snyderman and Hunt the domain of f was the collection of pairs of characters. The range of f was the collection of bytes, and f was invertible. The definitions of the Domain and Range for the other methods are summarized in table I.

It appears that compression techniques may be classified in terms of the type of domain, range and operation. Of the methods surveyed, the domain was composed of either fixed length or variable length elements. The range, except for those techniques that generate Huffman codes, was composed of fixed length elements. To generate Huffman codes, the function maps the domain into elements whose length is inversely proportional to the frequency of occurrence of the element in the domain.

In some cases the methods differ only in the function definition. The difference between the method of Snyderman and Hunt and the one for Huffman code with patterns is that in the first case the function maps characters and pairs into bytes while in the latter case the function maps these same elements into variable length fields.

Table I
Domain and Range of a Sample of Data Compresssion Methods

Method	Domain	Range
Snyderman & Hunt	pairs of characters	bytes
Schieber & Thomas	" " "	"
Jewell	" " "	bytes
Lynch	" " "	fixed length fields
Hahn	Characters	Three fields two are fixed length, other is multiple words
Ling & Palermo	fixed length fields	fixed length fields
Schuegraf & Heaps	text fragments	" " "
Huffman Code with patterns	pairs of characters	variable length binary strings

The performance of these methods, chosen somewhat arbitrarily to represent a cross sample of the data compression methods in the literature, differs both in terms of percent reduction and computation time. As one may suspect, the more complex methods, such as the Huffman code generators, require more computation time than the simpler methods like that of Snyderman and Hunt. The Huffman code method did obtain a greater percent reduction than the others, so the increased computation time may be worthwhile for some applications. On the other hand, the text fragment method of Schuegraf and Heaps takes a significantly longer computation time to accomplish roughly the same degree of compression as the simpler digraph methods. Table II contains a summary of the published performance of some data compression methods. Notice that the measure of performance in the table is the reduction of storage space. Later in the paper, that measure will be shown to be unreliable when compared to the measure of entropy of the data.

Table II
Published Results of Some Compression Techniques

Method	% Reduction	Data Base
Snyderman & Hunt[28]	35	Smithsonian Scientific Information Exchange 171,000,000 characters
JEWELL[15]	47	12000 char text
Schieber & Thomas[24]	43.5	40,000 bibliographic records average of each is 535 char total of 21,400,000 char
Lynch[19]	36 to 46	Institute of Elect. Eng. INSPECT system and British National Bibl. MARC system
Ling & Palermo[17]	50	not specified
Schuegraf & Heaps[26]	35	Marc Tapes, Issue 1
Huffman Code[10] with Patterns	62	Insurance Company Files

While the compression methods described in the Schuegraf and Heaps paper have limited utility, because, as noted above, their complexity does not increase their effectivness over the more simpler digraph methods, the discussion of variable length text fragments in that paper leads to a related question about the structure of the data base. what form should the dictionary take? Inverted-file retrieval systems using free text data bases commonly identify words as keys or index terms about which the file is inverted, and through which access is provided. The words of na-

tural language exhibit a Zipfian * rank-frequency relationship in which a small number of words account for a large proportion of word occurrences, while a large number of words occur infrequently. The inverted-file system involves large and growing dictionaries and thus may entail inefficient utilization of storage because of distribution characteristics. It may be advantageous to consider the formation of keys for file-inversion from units other than words. In particular if variable length text fragments are chosen as keys, then the above compression method may be a powerful method of conserving space in inverted-file systems. A related paper by Clare, Cook, and Lynch [4] discusses the subject of variable length text fragments in greater detail.

3.1 Noiseless Coding Problem

Most of the compression methods described in the literature are approximations to the solution of the noiseless coding problem, which is described as follows. A random variable takes on values $x_1,\ldots,x_m$ with probabilities $p_1,\ldots,p_m$ respectively. Code words $w_1,\ldots,w_m$ of lengths $n_1,\ldots,n_m$ respectively, are assigned to the symbols $x_1,\ldots,x_m$. The code words are combinations of characters taken from a code alphabet $a_1,\ldots,a_D$, of length D. The problem is to construct a uniquely decipherable code which minimizes the average code-word length $\bar{n} = \sum_1^M p_i n_i$. Such codes will be called optimal in this paper. Usually the alphabet consists of the symbols 0 and 1. The problem may be approached in three steps. First we establish a lower bound on $\bar{n}$; then we find out how close we can come to that lower bound; then we synthesize the best code. We shall indicate to what degree the various compression methods are attempts to synthesize the best code.

---

* The Zipf distribution is a hyperbolic distribution in which the probability of occurrence of a word is inversely proportional to the rank of the word. If r is the rank of a word, then the probability p is defined by $p(r) = \frac{k}{r}$; where k is a constant chosen so that the sum $\sum_1^N p(r_i) = 1$.

3.1.1 Uniquely Decipherable Codes. What is a uniquely decipherable code? For example, consider the following binary code:

x_1 0

x_2 010

x_3 01

x_4 10

The binary sequence 010 could correspond to any one of the three messages x_2, x_3x_1, or x_1x_4. Since the sequence 010 cannot be decoded accurately, the following definition is needed to establish a rule to avoid such sequences.

A code is uniquely decipherable if every finite sequence of code characters corresponds to at most one message.

One way to insure unique decipherability is to require that no code word be a prefix of another code word. If A, B, and C are finite sequences of code characters, then the juxtaposition of A and C, written AC, is the sequence formed be writing A followed by C. The sequence A is a prefix of the sequence B if B may be written as AC for some sequence C.

Codes which have the above property, namely that no code word is a prefix of another code word, are called instantaneous codes. The code below is an example of an instantaneous code.

x_1 0

x_2 100

x_3 101

x_4 11

Notice that the sequences 11111, 10101, or 1001 do not correspond to any message; so such sequences should never appear and can be disregarded. The commonly used ASCII and EBCDIC codes are also instantaneous; but they are such because of their fixed length, since all fixed length codes are instantaneous. Every instantaneous code is uniquely decipherable, but not conversely. To see this, for a given finite sequence of code characters of an instantaneous code,

proceed from left to right until a code word W is formed. If no such word can be formed, then the unique decipherability condition is vacuously satisfied. Since W is not the prefix of any code word, w must be the first symbol of the message. Continuing until another code word is formed, and so on, this process may be repeated until the end of the message.

The term instantaneous refers to the fact that the code may be deciphered step by step. If, when proceeding left to right, W is the first word formed, we know immediately that W is the first word of the message. In a uniquely decipherable code which is not instantaneous, the decoding process may have to continue for a long time before the identity of the first word is known. For example, if in the code

x_1 0
x_2 000000001
(n characters)

we received the sequence of n+1 characters 00....001 we would have to wait until the end of the sequence to find out that the first symbol is x_1. Fortunately, the solution to the noiseless coding problem can be realized with an instantaneous code. Notice that while the ASCII and EBCDIC codes are instantaneous, they are usually far from optimal.

3.1.2 Optimal Codes. The degree of the optimality of the code is measured by the entropy of the message or text. The entropy H(X) is defined as

$$H(X) = -\sum_{1}^{M} p_i \log_2 p_i$$

where $p_1,.....,p_M$ are the probabilities of the message symbols as defined in the above description of the noiseless coding problem.

The following theorem gives the lower bound to the average length $\bar{n}$ of the code.

(Noiseless Coding Theorem)[1]. If $\bar{n} = \sum_{1}^{M} p_i n_i$ is the average code word length of a uniquely decipherable code for the random variable X, then $\bar{n} \geq H(X)/\log D$, with equality if and only if $p_i = D^{-n_i}$. Note that H(X)/log D is the uncertainty of X using logarithms to the base D, that is,

$$\frac{H(X)}{\log_2 D} = -\sum_1^M p_i \frac{\log_2 p_i}{\log_2 D} = -\sum_1^M p_i \log_D p_i.$$

For the environment we are interested in, the coding alphabet is binary, so D = 2. Thus the lower bound is simply $\bar{n} \geq H(X)$. H(X) is not only the lower bound to the length of the code needed to represent the data, it also provides a measure of the improvement that may be expected by compressing the data. The comparison of the value of H(X) to the current average code size, which is 8 for ASCII or EBCDIC, gives a measure of the improvement that can be realized by compressing the data. If H(X)=8 then no compression is realizable by coding the data differently; if H(X)=5 then up to an 8 to 5 compression ratio may be obtained. The comparison of the improvement realized by a specific data compression technique to the theoretic improvement given by the above ratio can serve to evaluate the effectivness of the technique. The measure of effectivness usually given, the file length before and after compression, does not indicate the true level of compression, since the compression may have been due mainly to null suppression.

Any code that achieves the lower bound of the noiseless coding theorem is called absolutely optimal. The following code is an example of an absolutely optimal code.

X	Probabilities	Code Words
X_1	1/2	0
X_2	1/4	10
X_3	1/8	110
X_4	1/8	111

$$H(X) = \bar{n} = \frac{7}{4}$$

In a previous example of a Huffman code, figure 6, the average code length of the Huffman code was 1.7 bits per character, while the value of the entropy H(X) was 1.156 bits per character. That example illustrates the general impossibility of contructing an absolutely optimal code for arbitrary collections of characters. That example also illustrates that any coding method will be bound by the value of H(X).

3.2 Realization of Optimal Codes

While the theorem states the existence of an absolutely optimal code, in general the construction of one for an arbitrary set of probabilities is impossible. For a given set

of probabilities $p_1, \ldots\ldots, p_M$, if the code is to be absolutely optimal, the lengths of the code words must be chosen to satisfy $p_i = D^{-n_i}$ which is the same as

$$n_i = \frac{(-\log p_i)}{\log D}.$$

Obviously each n_i may not be an integer and yet satisfy the above condition. However we may do the next best thing by choosing the integer n_i to satisfy the inequalities:

$$\frac{-\log p_i}{\log D} \leq n_i < \frac{-\log p_i}{\log D} + 1$$

An instantaneous code can be shown to exist in which the code lengths satisfy the above inequality. The following theorem characterizes such codes.

Given a random variable X with uncertainty H(X), there exists a base D instantaneous code for X whose average code-word length $\bar{n}$ satisfies

$$\frac{H(X)}{\log D} \leq \bar{n} < \frac{H(X)}{\log D} + 1$$

For a proof see Ash, page 39.

This theorem says that the average code-word length may be made sufficiently small to be within one digit of the lower bound set by the noiseless coding theorem. That lower bound may be approached arbitrarily close if block coding is used. The success of the digram coding schemes is due to the fact that block coding of length 2 is used. Block coding works as follows. Instead of assigning a code word to each symbol x_i, we assign a code word to each group of s symbols. In other words, we construct a code for the random vector $Y = (X_1, X_2, \ldots\ldots, X_s)$, where the X_i are independent and each X_i has the same distribution as X. If each X_i assumes M values then Y assumes M^s values. The following example illustrates the decrease in the average code-word length by block coding.

X	p	Code Word
x_1	3/4	0
x_2	1/4	1

$\bar{n} = 1$

$Y = (X_1, X_2)$	p	Code Word
x_1x_1	9/16	0
x_1x_2	3/16	10
x_2x_1	3/16	110
x_2x_2	1/16	111

$\bar{n} = 9/16 + 3/16\ (2) + 1/4\ (3)$
$= 27/16$ code characters/2 values of X
$= 27/32$ code characters/value of X

By the above theorem, the average code-word length $\bar{n}_s$ for the block of length s satisfies

$$\frac{H(Y)}{\log D} \leq \bar{n}_s < \frac{H(Y)}{\log D} + 1 \text{ code characters/value of Y.}$$

$H(Y) = H(X_1,.....,X_s) \leq H(X_1)+.....+H(X_s)$ whether or not the X_i are independent from each other. If they are independent, then the inequality becomes an equality. If the X_i are identically distributed, then $H(X_1)+.....+H(X_s) = sH(X)$. In the classical case, both independence and identical distribution are assumed, in which case, the average code word length satisfies

$$\frac{sH(X)}{\log D} \leq \bar{n}_s < \frac{sH(X)}{\log D} + 1,$$

or

$$\frac{H(X)}{\log D} \leq \frac{\bar{n}_s}{s} < \frac{H(X)}{\log D} + \frac{1}{s}.$$

while for text files and messages, the independence of each X_i is a tenuous assumption, the assumption that each X_i is identically distributed is credible. Upon dropping the independence assumption the above inequality becomes

$$\frac{H(X_1,.....,X_s)}{s(\log D)} \leq \bar{n}_s < \frac{H(X)}{\log D} + \frac{1}{s}.$$

Thus we see that regardless of the independence of the elements of the block, the upper bound of the average code

length may be made as close to $\frac{H(X)}{\log D}$ as desired by increasing the block length. On the other hand, the lower limit may be smaller when the elements of the block are not independent as is the case frequently in text files. Thus for the conditions applicable to text files and messages the average code-word length may be made at least as small as the optimal length characterized by the noiseless coding theorem. The dependence of characters in text files may explain why the simple digraph methods are so successful. That dependence is further exploited in the method of Wagner which substitutes codes for entire English phrases.

3.3 Synthesis of the Huffman Code

So far only the existence of optimal codes has been discussed; now the synthesis of one such code, the Huffman code, will be illustrated. For the synthesis of optimal codes, only the instantaneous codes need to be considered since if a code is optimal with respect to the class of instantaneous codes, then it is also optimal with respect to all uniquely decipherable codes. This characteristic is indeed fortunate since instantaneous codes are the codes of choice for data transmission and processing applications. The precise statement of this characteristic is as follows.

If a code C is optimal within the class of instantaneous codes for the given probabilities $p_1, p_2, \ldots\ldots, p_M$, which means that no other instantaneous code for the same given set of probabilities has a smaller average code-word length than C, then C is optimal within the entire class of uniquely decipherable codes.

For a proof see Ash page 40.

An optimal binary code can be characterized by certain necessary conditions which restrict the choices of code lengths that may be assigned to each code. These characterizations are as follows.

Given a binary code C with word lengths $n_1, n_2, \ldots, n_M$ associated with a set of symbols with probabilities $p_1, p_2, \ldots\ldots, p_M$, assume, for convenience, that the symbols are arranged in order of decreasing probability ($p_1 \geq p_2 \geq \ldots\ldots \geq p_M$) and that a group of symbols with the same probability is arranged in order of increasing code-

word length. (If $p_i = p_{i+1} = \ldots = p_{i+r}$, then $n_i \leq n_{i+1} \leq \ldots \leq n_{i+r}$.) Then if C is optimal within the class of instantaneous codes, C must have the following properties:

a. Higher probability symbols have shorter code words, that is, $p_j > p_k$ implies $n_j \leq n_k$.

b. The two least probable symbols have code words of equal length, that is, $n_{M-1} = n_M$.

c. Among the code words of length n_M there must be at least two words that agree in all digits except the last. For example, the following code cannot be optimal since code

x_1 0
x_2 100
x_3 101
x_4 1101
x_5 1110

words 4 and 5 do not agree in the first three places.

For a proof see Ash page 41.

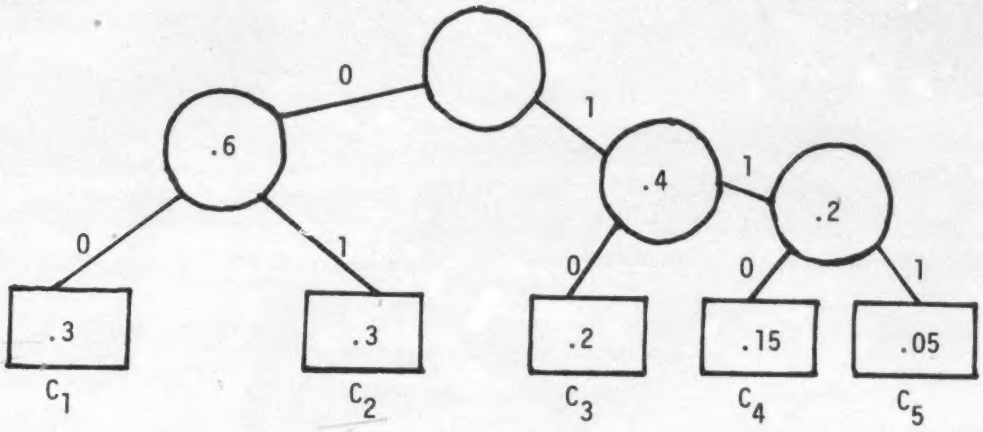
The construction of a Huffman code for the characters $c_1,\ldots,c_n$ with probabilities $p_1,\ldots,p_n$ respectively, involves generating a binary tree[1] for which each of the above characters is represented as a terminal node and the other nodes, the internal nodes, are formed in the following manner. First from the two nodes with smallest probabilities, say c_1 and c_2, a new node $c_{1,2}$ with probability p_1+p_2 is formed to be the father of c_1 and c_2. Now with the reduced set of n-1 nodes, which consists of $c_{1,2},c_3,\ldots,c_n$ with probabilities $p_1+p_2,p_3,\ldots,p_n$ respectively, repeat the above procedure; and continue to repeat it until reduced set consists of only two nodes. Now consider the binary tree which consists of the terminal nodes and all the new nodes formed by the above process. For each successive pairs of

---

[1] A binary tree is a graph which consists of a root node and descendent nodes. From the root node are links to at most two other nodes, the descendants of the root node. Each of these descendants, in turn, are linked to no more than two other nodes; and these latter nodes may be similarly linked to other nodes, and so on.

branches, starting at the root, assign the values 0 and 1 to each link of the branch. The resultant code for each of the characters is the sequence of assigned values obtained by tracing the tree from the root to each of the terminal nodes. Each aggregate causes the items so chosen to have a code length of one more binary digit; so the average length is minimized by giving this extra digit to the least probable clump. The following example illustrates the method.

Let the characters be c_1, c_2, c_3, c_4, c_5 and have probabilities .3, .3, .2, .15, .05, respectively. In the tree which results from the above method, the terminal nodes are represented by squares, the other nodes by circles, and in each square and circle is the probability of the node.

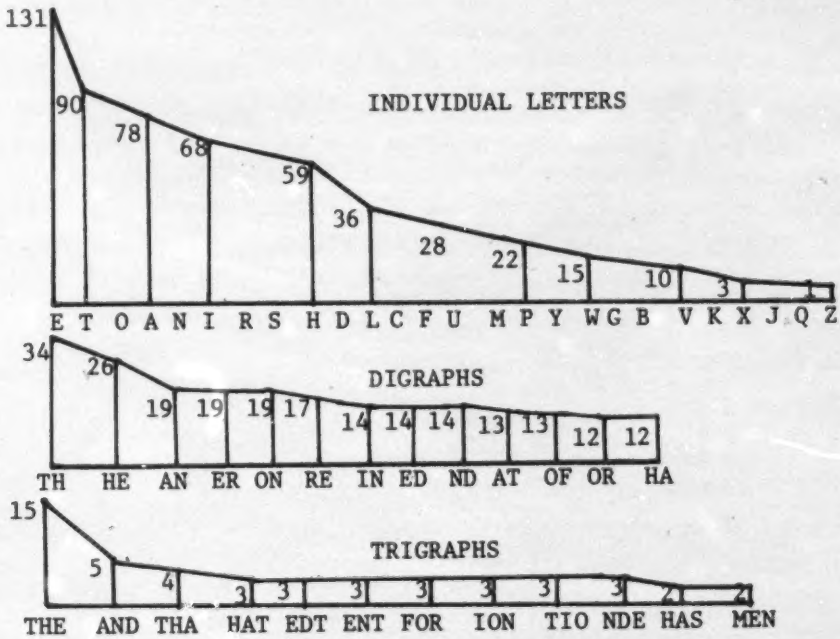


The Huffman code for each of the characters is:

Character	Code
c_1	00
c_2	01
c_3	10
c_4	110
c_5	111

A variation of the Huffman code, a variable length alphabetic code, is explained in a paper by Hu and Tucker. There, a tree, which is optimal in another sense, is obtained which preserves the original order of the terminal nodes. Using their algorithm, alphabetical codes may be generated which, though not as optimal as a Huffman code, enables ordering operations to be applied to the coded text in the same way as the uncoded text.

Observe that for the formation of the Huffman code the distribution of the characters or blocks must be known in advance. It may appear that the Huffman code is valid only for each instance or version of the data so that a new code may have to be generated for each data base and for each change to the data base. Fortunately, the distribution of characters is not that sensitive to changes in the data. One study has shown that the distribution of characters for a particular data base is stable over a period of time. [18] Moreover the same distribution seems to be relatively stable across different English text data bases. The following graph shows the distribution of characters in a typical English text.


INDIVIDUAL LETTERS
131 90 78 68 59 36 28 22 15 10 3 1
E T O A N I R S H D L C F U M P Y W G B V K X J Q Z
DIGRAPHS
34 26 19 19 19 17 14 14 14 13 13 12 12
TH HE AN ER ON RE IN ED ND AT OF OR HA
TRIGRAPHS
15 5 4 3 3 3 3 3 3 3 2 2
THE AND THA HAT EDT ENT FOR ION TIO NDE HAS MEN

Normal frequency distribution of the letters of the alphabet (in uses per thousand)

The following table, from the paper by Lynch, Petrie, and Snell [18], shows a distribution of characters which is close to that in the graph.

For a given Huffman code, changes in the average code word length with respect to changes in the distribution of the characters may be analyzed in the following way. Let the code word lengths be $n_1, n_2, \ldots\ldots, n_m$, where $n_1 \leq n_2 \leq \ldots\ldots \leq n_m$,

TABLE 1. NORMALIZED FREQUENCIES WITH MEANS AND STANDARD DEVIATIONS FOR THE FIRST 29 CHARACTERS (ARRANGED IN RANKED ORDER). THE FILES ANALYSED RANGE FROM INSPEC 31002 (1969) TO INSPEC 31060 (1972)

	31002	31003	31015	725	31016	710	31017	31056	31057	31060	S.D.	Mean
V	0·1511	0·1505	0·1488	0·1508	0·1332	0·1499	0·1504	0·1485	0·1498	0·1502	0·0054	0·1483
E	0·0889	0·0900	0·0885	0·0890	0·0890	0·0903	0·0900	0·0902	0·0906	0·0883	0·0039	0·0875
T	0·0730	0·0727	0·0729	0·0724	0·0725	0·0724	0·0722	0·0721	0·0719	0·0728	0·000360	0·0725
I	0·0725	0·0736	0·0755	0·0727	0·0741	0·0739	0·0738	0·0735	0·0736	0·0731	0·000856	0·0731
O	0·0722	0·0701	0·0705	0·0712	0·0705	0·0699	0·0695	0·0701	0·0693	0·0715	0·000915	0·0705
N	0·0677	0·0671	0·0677	0·0680	0·0672	0·0672	0·0678	0·0662	0·0669	0·0674	0·000527	0·0673
A	0·0641	0·0647	0·0659	0·0658	0·0645	0·0651	0·0644	0·0661	0·0658	0·0654	0·000712	0·0651
R	0·0568	0·0569	0·0565	0·0558	0·0572	0·0576	0·0571	0·0592	0·0573	0·0563	0·000949	0·0570
S	0·0530	0·0529	0·0542	0·0522	0·0537	0·0533	0·0535	0·0541	0·0537	0·0530	0·000608	0·0534
C	0·0398	0·0397	0·0401	0·0392	0·0402	0·0397	0·0403	0·0406	0·0394	0·0388	0·000545	0·0398
L	0·0370	0·0370	0·0379	0·0374	0·0378	0·0375	0·0370	0·0369	0·0370	0·0375	0·000368	0·0373
M	0·0267	0·0259	0·0271	0·0267	0·0267	0·0267	0·0247	0·0278	0·0257	0·0271	0·000870	0·0265
F	0·0259	0·0260	0·0261	0·0258	0·0259	0·0251	0·0262	0·0245	0·0262	0·0257	0·000540	0·0257
D	0·0248	0·0256	0·0256	0·0258	0·0261	0·0252	0·0259	0·0258	0·0258	0·0254	0·000380	0·0256
U	0·0238	0·0238	0·0238	0·0233	0·0240	0·0231	0·0234	0·0232	0·0240	0·0237	0·000331	0·0236
H	0·0222	0·0226	0·0208	0·0218	0·0216	0·0220	0·0222	0·0214	0·0224	0·0217	0·000529	0·0219
P	0·0220	0·0210	0·0217	0·0218	0·0215	0·0223	0·0213	0·0227	0·0215	0·0224	0·000531	0·0218
G	0·0156	0·0156	0·0159	0·0146	0·0160	0·0162	0·0151	0·0165	0·0155	0·0153	0·000554	0·0156
Y	0·0126	0·0129	0·0125	0·0124	0·0122	0·0122	0·0123	0·0119	0·0125	0·0127	0·000310	0·0123
B	0·0086	0·0089	0·0087	0·0092	0·0086	0·0085	0·0089	0·0084	0·0090	0·0090	0·000257	0·0089
V	0·0071	0·0072	0·0071	0·0071	0·0076	0·0076	0·0073	0·0078	0·0074	0·0069	0·000285	0·0073
–	0·0061	0·0063	0·0063	0·0064	0·0063	0·0063	0·0062	0·0061	0·0064	0·0060	0·000137	0·0063
W	0·0052	0·0055	0·0056	0·0048	0·0057	0·0056	0·0053	0·0056	0·0056	0·0050	0·000303	0·0054
X	0·0026	0·0030	0·0026	0·0031	0·0025	0·0027	0·0027	0·0025	0·0027	0·0028	0·000199	0·0027
K	0·0022	0·0022	0·0021	0·0022	0·0022	0·0022	0·0023	0·0023	0·0023	0·0023	0·000067	0·0022
·	0·0020	0·0021	0·0021	0·0020	0·0022	0·0020	0·0023	0·0019	0·0019	0·0017	0·000169	0·0020
•	0·0019	0·0016	0·0015	0·0019	0·0015	0·0017	0·0017	0·0013	0·0013	0·0020	0·000246	0·0016
Q	0·0018	0·0019	0·0018	0·0017	0·0019	0·0020	0·0018	0·0019	0·0018	0·0017	0·000095	0·0018
Z	0·0016	0·0015	0·0015	0·0016	0·0015	0·0015	0·0014	0·0015	0·0016	0·0017	0·000084	0·0015

and the probabilities of the characters are $p_1, p_2, \ldots, p_m$. Suppose that the i'th probability changes by the amount d_i, so that $\hat{p}_i = p_i + d_i$ is the new i'th probability. The new average code word length is

$$\bar{n}' = \sum_1^m \hat{p}_i n_i = \sum_1^m (p_i + d_i) n_i = \bar{n} + \sum_1^m d_i n_i .$$

Let $D = \sum_1^m d_i n_i$. Then since $\sum_1^m d_i = 0$, $D = \sum_1^{m-1} d_i (n_i - n_m)$. There are two interesting cases to consider. The first occurs when $d_i \geq 0$ for $i=1,2,\ldots,m-1$. Then, since $n_i - n_m \leq 0$, $D \leq 0$ so $\bar{n}' \leq \bar{n}$. The second case occurs when $d_i \leq 0$ for $i=1,2,\ldots,m-1$. Then $\bar{n}' \geq \bar{n}$. If the changes d_i are restricted so that

$$|d_i| \leq \frac{a^i}{n_m - n_i}$$

then

$$D = \sum_1^{m-1} (-d_i)(n_m - n_i) < \sum_1^{m-1} a^i = \frac{a(1-a^{m-1})}{1-a} .$$

If $a \leq \frac{1}{2}$ then $D < 1 - (\frac{1}{2})^{m-1} < 1$. It appears that as long as the distribution of characters changes only slightly, from data base to data base, a Huffman code designed for one of the data bases will be adequate for the others. Further study of the variation of Huffman codes with respect to changes in the data base is needed before more detailed statements can be made about the performance of Huffman codes when such changes occur.

4. CONCLUSIONS

Several types of compression methods have been discussed along with the underlying coding theory and the measures for evaluating the effectiveness of a compression method. It was shown that the data compression problem is the same as the optimal coding problem when the data file is considered as a collection of independent characters. Since data characters are generally not independent, the optimal code may be even shorter than that predicted by the noiseless coding theorem, thus possibly permitting even greater compression. A good measure of the effectiveness of the method is not the percent reduction, used in some of the referenced papers, but the ratio of the entropy H(x) of the data file to the average encoded character size in bits. If the compression is at least as good as the optimal code then the ratio is greater than or equal to 1, otherwise it is less than one.

The steps to be followed in selecting or determining a need for a data compression method involve the calculation of the entropy of the data. These steps are:

1. Measure H(X), where

$$H(X) = \sum_{i=1}^{N} p_i \log_2(p_i).$$

In the above formula for H(X), $p_i = f_i/F$, where f_i is the frequency of the i'th type of element of the data file, and F is the total number of elements in the file ($F = \sum_{i=1}^{N} f_i$), and N is the number of distinct types of elements. As in section 3.1, the data file is composed of a sequence of elements which are usually characters. In ASCII data files, there are 128 different types of characters that may occur in the file; however, since control characters usually do not occur in a file, most ASCII files will have only 96 possible types of characters. Alternatively H can be calculated from the equivalent expression

$$H(X) = (1/F) \sum_{i=1}^{N} f_i \log_2(f_i) - \log_2(F)$$

by summing the values $f*\log_2(f)$ for each character, dividing

by F and then subtracting $\log_2(F)$. For large data files, it is not necessary to base the calculations on the entire file, but only on part of the file, say the first 100,000 bytes if the file is homogeneous, or one can use some random sampling procedure to estimate the frequencies f_i.

2. Determine the current average character length $\bar{n}$ in bits. For ASCII and EBCDIC files this value will usually be 8. If H(X) is much less than $\bar{n}$ then a statistical compression method will be effective. If, on the other hand, H(X) is close to $\bar{n}$ then such methods will not be effective; however some type of pattern substitution may be applicable. For example, if H(X)=7 and the current code-word length is 8 then some improvement would be expected by compressing the data, but, on the other hand a greater improvement is to be expected when H(X)=5 and the current length is 8.

3. If the data is numerical, then a numerical method such as polynomial predictors and polynomial curve fitting algorithms [5-9] may be superior to the methods discussed in this report.

4. If the data is text or a combination of text and numerical tables, and the data is compressible as indicated in step 2, then either a digraph method or a Huffman method would compress the data. The digraph method is much easier to implement, and runs faster than the Huffman method, while the latter obtains a higher degree of compression. The choice of the compression method will depend on the characteristics and applications of the data. Data files which contain mostly numeric fields would be compressible by an entirely different algorithm than would text files. Frequently accessed files may need an algorithm which runs quicker than that for less frequently accessed files, even though the data compression obtained by the faster algorithm is far less then optimal. Within the same file system parts of the file may be more efficiently compressed with different methods. The dictionary* of an information management system may be compressed with a simple yet fast algorithm, while the corresponding data files, because they are infrequently accessed, may be compressed with a more complex al-

---

* The dictionary as used here, refers to the collection of pointers of an inverted file system. Each pointer, by pointing to a record of the file, functions in a manner analogous to a word of an English language dictionary.

gorithm which is slower but realizes more compression. A variable length alphabetic code**, which has some of the optimal properties of the Huffman code, may be used to compress the dictionary.

5. The effectiveness of a particular data compression method can be measured by comparing the average character length of the data file after it has been compressed to the value of the entropy of the file. If the average character length, after compression, is close to the value of the entropy then the method is as effective as an optimal statistical compression method. If the value of the average is still significantly greater than the value of the entropy, then the data compression method is not as effective as possible.

Data compression is relevant to a data processing application when its use is significant or meaningful to the user. Its use is warranted when it effects at least one of the following:

1. Significant cost reduction
2. Significant storage reduction
3. Allowing the implementation of the application which otherwise could not have been implemented due to insufficient storage
4. A significant decrease in the data transfer time.

The notion of what is significant to a user is relative to the users environment. To a mini-computer user with limited disc storage, a reduction of a few thousand bytes of storage may be significant, while to a large system user such a reduction would be insignificant. While the ultimate decision of whether or not data compression is relevant depends on the users special requirements and judgement, the following three guidelines will be applicable in most cases.

1. If the quantity of data is small, say under 100,000 bytes, or if the life of the data is short, then data compression would not be advisable.
2. Large data files, over 100,000 bytes, the life of which is not short, are good candidates for data compression.
3. A group of data files, where the files have similar character composition, is a good candidate for data compression when the size of the group is more than 100,000 bytes.

** see section 3.3

5. BIBLIOGRAPHY

1. Ash,Robert; "Information Theory"; Interscience 1965
2. Barton,I.J., Creasey,S.E., Lynch,M.F.,and Snell,M.J. "An Information-theoretic Approach to Text Searching in Direct Access Systems". Comm. of ACM. 17,6 (June 1974) pp 345-350.
3. Bookstein,A., and Fouty,G. "A Mathematical Model for Estimating the Effectiveness of Bigram Coding". Int. Proc. and Manag. 12,(1976) pp 111-116.
4. Clare,A.G., Cook,G.M., and Lynch,M.F. "The Identification of Variable-length, Equi-frequency Character Strings In A Natural Language Data Base". Computer J. 15 (1972)
5. Davisson, L.D. "An Approximate Theory of Prediction for Data Compression", IEEE Trans.,IT-13,No 2,(1967) pp 274-278.
6. Davisson, L.D. "Data Compression Using Straight Line Interpolation", IEEE Trans.,It-4, No. 3,(1968) pp 390-304.
7. Davisson, L.D. "The Theoretical Analysis of Data Compression Systems", Proc. IEEE, Vol 56, No. 2,(1968) pp 176-186.
8. Ehrman,L. "Analysis of Some Redundancy Removal Bandwidth Compression Techniques", Proc. IEEE, Vol 55, No.3,(1967) pp 278-287.
9. Ellias,P. "Predictive Coding". IRE Trans., IT-1,(1955) pp 16-33.
10. Gottlieb,B., Hagereth,S.E., Lehot,P.G.H., and Rabinowitz,H.S. "A Classification of Compression Methods and Their Usefulness For A Large Data Processing Center". Proc. National Comp. Conf., 1975, pp 453-458.
11. Hahn,B. "A New Technique For Compression and Storage of Data". Comm. ACM. 17,8 (1974).
12. Heaps,H.S. "Storage Analysis of A Compression Coding For Document Data Bases". Infor,10,1 (Feb 1972).
13. Hu,T.C., and Tucker,A.C. "Optimal Computer Search Trees and Variable-Length Alphabetical Codes". SIAM J. Appl. Math. Vol 21,4 (Dec 1971) pp 514-532
14. Huffman,D. "A Method For The Contruction of Minimum Redundency Codes". Proc. I.R.E. 1953,40, pp 1098-1104.
15. Jewell,G.C. "Text Compaction For Information Retrieval Systems". IEEE SMC Newsletter, 5,1 (Feb 1976).
16. Lesk,M.E. "Compressed Text Storage". Computing Science Technical Report #3. Bell Telephone Laboratories 1970.
17. Ling,H. and Palermo,F.P. "Block-orientated Information Compression". IBM J. RES. DEVELOP., March 1975.

18. Lynch,F.L.,Petrie,H.J., and Snell,M.J. "Analysis of The Microstructure of Titles In The Inspec Data-base". Inform. Stor. Retr. Vol 9, (1973) pp 331-337
19. Lynch,M.F. "Compression of Biliographic Files Using An Adaption of Run-length Coding". Inf. Stor. Retr., 9 (1973) pp 207-214.
20. Marron, B.A. and De Maine, P.A.D., "Automatic Data Compression", Comm. ACM. vol 10, 3 (Nov 1967) pp 711-715.
21. Mayne,A. and James,E.B., "Information Compression By Factorising Common Strings". The Computer Journal, 18,2, pp 157-160.
22. McCarthy,J.P. "Automatic File Compression". Proc. Intern. Comp. Symp., 1973, North-Holland, pp 511-516.
23. Ruth,S., and Kreutzer,P. "Data Compression For Large Business Files". Datamation ,18, 9 (sept 1972).
24. Schieber,W.D., and Thomas,G. "An Algorithm For The Compaction of Alphanumeric Data". J. Library Autom. 4,(1971) pp 198-206.
25. Schuegraf,E.F., and Heaps,H.S. "Selection of Equifrequent Word Fragments For Information Retrieval". Inform. Stor. Retr. Vol 9 (1973), pp 697-711.
26. Schuegraf,E.J., and Heaps, H.S. "A Comparison of Algorithms For Data Base Compression By The Use of Fragments As Language Elements". Infor. Stor. Retr.,10 (1974) pp 309-319.
27. Shannon,C.E. "A Mathematical Theory of Communication". Bell Syst. Tech. J. 1948,27
28. Snyderman,M. and Hunt,B. "The Myriad Virtues of Text Compaction". Datamation. 16,(Dec 1,1970) pp 36-40.
29. Thiel,L.H., and Heaps,H.S. "Program Design For Retrospective Searches On Large Data Bases". Inform. Stor. Retr., Vol 8 (1972), pp 1-20.
30. Wagner, Robert A., "Common Phrases and Minimum-Space Text Storage", Comm. ACM., Vol 16, 3 (March 1973) pp 148-152

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS SP 500-12	2. Gov't. Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

Data Compression - A Comparison of Methods

5. Publication Date: June 1977

6. Performing Organization Code

7. AUTHOR(S)

Jules Aronson

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No. *640-1225*

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address (*Street, City, State, ZIP*)

Same as item 9

13. Type of Report & Period Covered *Final*

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 77-608132

16. ABSTRACT (*A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.*)

One important factor in system design and in the design of software is the cost of storing data. Methods that reduce storage space can, besides reducing storage cost, be a critical factor in whether or not a specific application can be implemented. This paper surveys data compression methods and relates them to a standard statistical coding problem -- the noiseless coding problem. The well defined solution to that problem can serve as a standard on which to base the effectiveness of data compression methods. A simple measure, based on the characterization of the solution to the noiseless coding problem, is stated through which the effectiveness of a data compression method can be calculated. Finally, guidelines are stated concerning the relevence of data compression to data processing applications.

17. KEY WORDS (*six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons*)

Coding; coding theory; computer storage; data compaction; data compression; data elements; data management; data processing; information management; information theory.

18. AVAILABILITY

[X] Unlimited

[ ] For Official Distribution. Do Not Release to NTIS

[X] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13.10:500-12

[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151

19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
UNCLASSIFIED	39
20. SECURITY CLASS (THIS PAGE)	22. Price
UNCLASSIFIED	

USCOMM-DC 29042-P74